LANGLEY GRANT
IN-61-CR
90658
14P.

Annual Progress Report
Award No. NAG-1-605
July 1, 1985 - June 30, 1987

DETECTION OF FAULTS AND SOFTWARE RELIABILITY ANALYSIS

Submitted to:

National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia 23665

Attention: Mr. Gerard E. Migneault
FCSD M/S 130

Submitted by:

J. C. Knight
Associate Professor

(NASA-CR-180347) DETECTION OF FAULTS AND SOFTWARE RELIABILITY ANALYSIS Annual Progress Report, 1 Jul. 1985 - 30 Jun. 1987 (Virginia Univ.) 14 p Avail: NTIS HC A02/MF A01 CSCL 09B G3/61
N87-27433
Unclas
0090658

Report No. UVA/528243/CS88/103
August 1987

SCHOOL OF ENGINEERING AND APPLIED SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF VIRGINIA
CHARLOTTESVILLE, VIRGINIA 22901

Annual Progress Report
Award No. NAG-1-605
July 1, 1985 - June 30, 1987

DETECTION OF FAULTS AND SOFTWARE RELIABILITY ANALYSIS

Submitted to:

National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia 23665

Attention: Mr. Gerard E. Migneault
FCSD M/S 130

Submitted by:

J. C. Knight
Associate Professor

Department of Computer Science
SCHOOL OF ENGINEERNG AND APPLIED SCIENCE
UNIVERSITY OF VIRGINIA
CHARLOTTESVILLE, VIRGINIA

Report No. UVA/528243/CS88/103 Copy No. 4
August 1987

TABLE OF CONTENTS

SECTION I

INTRODUCTION

The work carried out under this grant has been an investigation of software faults in a number of areas. The goal was to better understand their characteristics and to apply this understanding to the software development process for crucial applications in an effort to improve software reliability. Some of the work was empirical and some analytic., The empirical work was based on the results of the Knight and Leveson experiment [1] on *N*-version programming. The analytic work has built useful models of certain aspects of the software development process.

Multi-version or *N*-version programming [2] has been proposed as a method of providing fault tolerance in software. The approach requires the separate, independent preparation of multiple (i.e. ''*N*'') versions of a piece of software for some application. These versions are executed in parallel in the application environment; each receives identical inputs and each produces its version of the required outputs. The outputs are collected by a voter and, in principle, they should all be the same. In practice there may be some disagreement. If this occurs, the results of the majority (assuming there is one) are taken to be the correct output, and this is the output used by the system.

The major experiment carried out by Knight and Leveson was designed to study *N*-version programming and initially investigated the assumption of independence. In the experiment, students in graduate and senior level classes in computer science at the University of Virginia (UVA) and the University of California at Irvine (UCI), were asked to write programs from a single requirements specification. The result was a total of twenty-seven programs (nine from UVA and eighteen from UCI) all of which should produce the same output from the same input.

Each of these programs was then subjected to one million randomly-generated test cases. The Knight and Leveson experiment has yielded a number of programs containing faults that are useful for general studies of software reliability as well as studies of *N*-version programming.

Our work has been in a number of areas and each area is covered separately in this report. The specific topics are:

(1) the Consistent Comparison Problem in *N*-version systems,

(2) analytic models of comparison testing,

(3) fault tolerance through data diversity,

(4) and the relationship between failures caused by automatically seeded faults.

This report is quite brief since the details of the research have been reported in published or submitted papers. These papers have been supplied to the sponsor separately and are merely referenced here.

SECTION II

CONSISTENT COMPARISON

We have identified previously a difficulty in the implementation of *N*-version programming. The problem, which we call the *Consistent Comparison Problem*, arises for applications in which decisions are based on the results of comparisons of finite-precision numbers. We have shown that when versions make comparisons involving the results of finite-precision calculations, it is impossible to guarantee the consistency of their results. It is therefore possible that correct versions may arrive at completely different outputs for an application that does not apparently have multiple correct solutions. There is no solution to the Consistent Comparison Problem, and we have been unable to find techniques for avoiding it. If this problem is not dealt with explicitly, an *N*-version system may be unable to reach a consensus even when none of its component versions fail.

A revised paper describing this work has been submitted to the IEEE Transactions on Software Engineering [3]. The paper was revised based on the comments made by referees on a previous version of the paper.

SECTION III

COMPARISON TESTING

A common argument [4] in favor of at least dual programming (i.e. N-version programming with $N = 2$) is that testing of safety-critical real-time software can be simplified by producing two versions of the software and executing them on large numbers of test cases without manual or independent verification of the correct output. The output is assumed correct as long as both versions of the programs agree. The argument is made that preparing test data and determining correct output is difficult and expensive for much real-time software. Since it is assumed ''unlikely'' that two programs will contain identical faults, a large number of test cases can be run in a relatively short time and with a large reduction in effort required for validation of test results. We refer to this approach as *comparison testing* although it is also known as back-to-back testing in the literature.

Comparison testing has been criticized on the grounds that it tends to reveal only those faults for which the programs generate different outputs. Such faults are inconvenient but not dangerous to an N-version system since they will be detected and tolerated. Comparison testing will not reveal faults that cause identical wrong outputs and it is precisely these that will not be tolerated.

We have found that comparison testing is a very useful and cost-effective method of fault elimination in multi-version systems. The reason is that although two faults in different programs may cause coincident failures, our experience has been that such faults do not *always* cause coincident failures. Thus there are occasions when only one of the two programs will fail allowing comparison testing to detect the situation.

We have analyzed the performance of comparison testing from several viewpoints. First, we have built Markov models of the fault location process. The models associate states with the number of located faults and the order in which they are found. Transition probabilities between states are just the probabilities of finding particular faults on each test case. The expected number of tests to locate each fault even for faults that cause coincident failures can be determined from such models. The models show that comparison testing is remarkably effective when compared with an ideal (and unrealizable) testing system employing an oracle.

In this work we have also derived cost models of comparison testing, and models of the reliability that will be exhibited by N-version systems built from versions developed using comparison testing.

This work is presented in a PhD dissertation [5] that has been supplied separately to the sponsor.

SECTION IV

FAULT TOLERANCE THROUGH DATA DIVERSITY

We have proposed a new approach to software fault tolerance that we term *data diversity*. Fault tolerance has been attacked in the past through design diversity. We suggest that it might be achieved through diversity in the data.

The basis of the approach is to execute several copies of a single program but supply each with different data. The sets of data, although different, are semantically equivalent. The idea of executing multiple copies of a single version of software has been rejected by others as pointless. The argument for rejection is that if one copy fails they will all fail. We have found, however, that executing several copies with different but equivalent inputs is a useful approach.

In fact, a variant of this approach has been suggested and tried by industrial software developers. Their approach is to use conventional *N*-version programming but to stagger the times at which the versions read sensors so that they will each receive slightly different data values. In practice, it is not necessary or even beneficial to use different versions and it is not necessary to await changes in the data over time. The changes can be computed.

There have been no previous analyses or experiments performed to evaluate the performance of either the industrial approach or our method of data diversity. We have performed analytic and simulations studies of both and have very encouraging results.

This work was reported in a paper presented at the Seventeenth International Symposium on Fault Tolerant Computing [6] and a more detailed report has been submitted to the IEEE Transactions on Computers [7].

SECTION V

SEEDED FAULTS

In the *N*-version programming method, separate development is intended to eliminate the sharing of (mis)understanding of the application; it associates independence of program failures with mutual isolation of the program designs. However, separate development has no effect on errors unbiased by knowledge of the application. For example, a programmer may inadvertently misorder certain steps in a computation or reverse the use of ''and'' and ''or'' in a conditional expression. The important characteristic of these errors is that they are not specific to the application. The separate development process does not affect their introduction. We have examined whether unbiased errors play any role in the expected independence of the resulting programs.

We have adopted an operational definition of independence: failures of two programs are dependent if a statistical measure shows a correlation of incorrect outputs for a given input. That is, programs that fail together significantly more often than expected are considered to contain dependent errors. How dependent errors are introduced does not affect the operational viewpoint of independence*. The statistical measure used here, a χ^2 test of an independence hypothesis, is the same as has been applied in [1]. A hypothesis that two programs fail independently is formed and the χ^2 statistic is generated. When the hypothesis is rejected with a high confidence level, dependence is assumed.

As part of a separate project we have performed an experiment in error seeding. Seventeen of the twenty-seven programs produced in the Knight and Leveson experiment were selected at random, errors were seeded into all seventeen, and the resulting programs were tested. The

* We note that other authors use different definitions of independence, for example [8].

algorithms used for seeding errors were very simple: 2 algorithms modified the bounds on **for** statements, 3 algorithms modified the Boolean expression in **if** statments, and 1 algorithm deleted assignment statements. Each of these algorithms was applied 4 times to each of the 17 programs for a total of 408 modified programs, each of which contained one seeded error. It should be stressed that the seeded errors were introduced at random without using any semantic knowledge of the program structure. To introduce one seeded error, a syntactic structure was selected at random and the seeding algorithm was applied. The seeded errors are unbiased errors.

To select seeded errors to be investigated for dependent failures a form of acceptance testing was used: seeded errors with a mean time to failure smaller than a certain threshold were disqualified from the experiment. In addition, seeded errors which caused *no* failures during the original error seeding experiment were also disqualified. 45 of the 408 seeded errors passed this acceptance test. Such an acceptance test is equivalent to the original acceptance testing done to admit the launch interceptor programs to the original *N*-version experiment. In this experiment all indigenous errors were fixed before the seeded errors were installed in the programs. Each failure of a given program is caused only by the seeded error.

The results of this experiment showed that seeded errors tended to cause coincident failures at a rate far higher than would be expected by chance. This is surprising, and its effects need to be taken into account in determining the reliability of *N*-version systems. This work has been documented in a paper that has been submitted to the IEEE Transactions on Software Engineering [9].

REFERENCES

(1) J.C. Knight and N.G. Leveson, ''An Experimental Evaluation Of The Assumption Of Independence In Multi-Version Programming,'', *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 1, January 1986.

(2) L. Chen and A. Avizienis, ''*N*-Version Programming: A Fault-Tolerance Approach To Reliability Of Software Operation'', *Digest of Papers FTCS-8: Eighth Annual International Conference on Fault Tolerant Computing*, Toulouse, France, pp. 3-9, June 1978.

(3) S.S. Brilliant, J.C. Knight, and N.G. Leveson, ''The Consistent Comparison Problem In *N*-Version Software'', *IEEE Transactions on Software Engineering*, to appear.

(4) C.V. Ramamoorthy, Y.R. Mok, E.B. Bastani, G.H. Chin, and K. Suzuki. ''Application Of A Methodology For The Development And Validation Of Reliable Process Control Software,'', *IEEE Transactions on Software Engineering*, vol. SE-7, no. 6, pp. 537-555, Nov. 1981.

(5) S.S. Brilliant, ''Testing Multi-Version Software'', Ph.D. Dissertation, University of Virginia, September, 1987.

(6) P.E. Ammann and J.C. Knight, ''Data Diversity: An Approach To Software Fault Tolerance'', *Digest of Papers FTCS-17: Seventeenth Annual International Conference on Fault Tolerant Computing*, Pittsburgh, PA, pp. 122-126, July 1987.

(7) P.E. Ammann and J.C. Knight, ''Data Diversity: An Approach To Software Fault Tolerance'', submitted to *IEEE Transactions on Computers*.

(8) A. Avizienis "The N-Version Approach To Fault Tolerant Software", *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 12 (December 1985).

(9) P.E. Ammann and J.C. Knight, "The Failure Characteristics of Unbiased Faults and Their Relation to Multi-Version Software", submitted to *IEEE Transactions on Software Engineering*.

DISTRIBUTION LIST

Copy No.	
1 - 3	National Aeronautics and Space Administration Langley Research Center Hampton, Virginia 23665 Attention: Mr. Gerard E. Migneault FCSD M/S 130
4 - 5*	NASA Scientific and Technical Information Facility P.O. Box 8757 Baltimore/Washington International Airport Baltimore, Maryland 21240
6 - 7	J. C. Knight, CS
8	R. P. Cook, CS
9 - 10	E. H. Pancake, Clark Hall
11	SEAS Publications Files

*1 reproducible copy

0239:ald:DW209R

UNIVERSITY OF VIRGINIA
School of Engineering and Applied Science

The University of Virginia's School of Engineering and Applied Science has an undergraduate enrollment of approximately 1,500 students with a graduate enrollment of approximately 560. There are 150 faculty members, a majority of whom conduct research in addition to teaching.

Research is a vital part of the educational program and interests parallel academic specialties. These range from the classical engineering disciplines of Chemical, Civil, Electrical, and Mechanical and Aerospace to newer, more specialized fields of Biomedical Engineering, Systems Engineering, Materials Science, Nuclear Engineering and Engineering Physics, Applied Mathematics and Computer Science. Within these disciplines there are well equipped laboratories for conducting highly specialized research. All departments offer the doctorate; Biomedical and Materials Science grant only graduate degrees. In addition, courses in the humanities are offered within the School.

The University of Virginia (which includes approximately 2,000 faculty and a total of full-time student enrollment of about 16,400), also offers professional degrees under the schools of Architecture, Law, Medicine, Nursing, Commerce, Business Administration, and Education. In addition, the College of Arts and Sciences houses departments of Mathematics, Physics, Chemistry and others relevant to the engineering research program. The School of Engineering and Applied Science is an integral part of this University community which provides opportunities for interdisciplinary work in pursuit of the basic goals of education, research, and public service.